Amendments to the Specification

Please amend the title as follows:

A DISPLAY SYSTEM, COMPUTER-READABLE STORAGE DEVICE, AND METHOD FOR ~~DYNAMIC~~ DYNAMICALLY ENABLING AND DISABLING BUFFERING OF ~~GRAPHIC~~ IMAGES ~~BY A PLATFORM INDEPENDENT APPLICATION PROGRAM INTERFACE~~

Please amend or delete the paragraphs beginning on page 10, line 26, as follows:

~~Furthermore,~~ For example, because Swing buffers its output, ~~when several Swing components are contained within each other,~~ performance can suffer (relative to an equivalent AWT implementation) when several Swing components are contained within each other.

~~Also, Swing does not allow the association of look and feel characteristics with a particular thread. This makes it impossible for an application to have its own unique look and feel, while the operating system under which the application is running maintains the global look and feel of the platform.~~

~~In addition, the architecture of the JTextField and JPasswordField components in Swing is different from that of the TextField AWT component. Unless modified, these Swing components may not function properly when used by the legacy application. Moreover, Swing TextField and Label component Peers often exhibit poorer performance than their AWT counterparts.~~

~~A further consideration when evaluating Swing as an API for a portable GUI is the limited functionality of Swing. Swing lacks some of the functionality available in the better native GUIs (such as Windows®). For example, the Windows operating system provides the capability for application programs using text components to popup menus (e.g., for help-related functions) with undo/redo and editing support. This capability is absent from Swing text~~

~~components, as well as the legacy AWT. Therefore, there is a need for a platform-independent replacement for the AWT that overcomes the above-mentioned limitations of Swing.~~

In view of these, and other considerations, it would be desirable to have some means of replacing the heavyweight AWT Peer classes with lightweight Peers (such as those in Swing), while avoiding the problems and limitations described above.

~~The~~ An API having both Swing-type features and AWT-type features is ~~alternatively called~~ referred to herein as "AWTSwing," and can be used to provide improved functionality and a more consistent look and feel for Java application programs. AWTSwing replaces the heavyweight Peer classes in the AWT with lightweight Peer classes while preserving the same interface. This allows legacy applications, which previously relied upon the graphics resources of the AWT, to run without modification. AWTSwing also augments the functionality of Swing, so that new applications can deploy a modern full-featured graphical interface. Because AWTSwing uses no native code for the windowing controls, it is truly portable. Therefore, the GUI for a Java application based on AWTSwing will have a consistent look and feel regardless of which operating system it is ported.

~~According to one embodiment, a display system is contemplated. The display system is one applicable to a computer system, and more particularly includes a display (i.e., computer monitor), a graphical user interface, and a processor. The processor can be adapted to operate from an operating system, such as a Windows-based operating system. The processor is coupled to not only the operating system, but also a software component, during run-time of an application program. Thus, the application program initiates a software component. Alternatively, the software component can be considered as a sequence of code of the application program, such code being that which produces an image accessible by a user via the graphical user interface. When executing the software component by the processor, the processor generates a first image upon the display independent of code within the operating system during a first time. During a second time, the executed software component emulates code that generates a second image upon the display dependent of code within the operating~~

~~system. Thus, the software component is contained within an API as either having its own native code dependent on the operating system, or simply having a template which can borrow from a parent software component contained within a plurality of Swing components. In the former instance, the software component has code unique to the operating system, but in the latter instance, the software component has code entirely separate from the operating system. More particularly, the software component in the latter instance, would borrow code from an ancestor, Swing component contained within a library of Swing components separate and apart from, and not dependent in any way upon, the operating system or platform on which the processor will run. Advantageously, the software component contains an object which will have the same look and feel when executed by the processor and displayed on the display device, regardless of the operating system. Thus, the buttons, list boxes, scroll bars, icons, colors, shading, etc. of the image will look substantially the same regardless of whether the API is using a platform specific API (AWT-type API) or a platform independent API (Java Swing-type API). Furthermore, the software component can utilize an AWT-type API or a Swing-type API either in combination, or separately. For example, it is possible for two sub-components to use two different APIs, or for all sub-components to use a single API at one time and a different API at another time.~~

~~According to another embodiment, a method is provided. The method is one which displays an image on a computer display. The computer can run an application program that includes an API dependent on the operating system of the computer to cause a first image to be displayed using the first interface. Thereafter, the computer can run an application program with a second interface that is substantially independent of the operating system, yet emulates the behavior of at least a part of the first interface. The application program can then be re-run to re-display the second image having substantially the same look and feel of the first image.~~

~~According to yet another embodiment, a computer readable storage device is presented. The storage device includes computer-executable code, such as an operating system and an application program. The application program can execute code of a software component during a first time and during a second time. During the first time, the code is executed by generating a first image independent of executing code within the operating system. During the second time,~~

~~code of the software component is executed to generate a second image dependent on executing code within the operating system, where the first image can overwrite an image upon a display screen previous to the first image. However, the first image cannot overwrite the second image upon the display screen during the first time. In other words, the software component cannot execute an AWT-type API, and map the software component outcome thereof to a Swing-type API. Instead, either the AWT-type API is executed or the Swing-type API is executed, at two dissimilar times, not in series with one another.~~